ROWS Pitch Angle and Scanning Antenna Pointing Angle Controller

By Steven A. Bailey

NASA
December 22, 1998

Table of Contents

Introduction 3

History 3

Initial Work...3

Testing The Initial Openloop Stepper Motor Design 4

New Design For Openloop Operation 4

Controller Design 5

Angle Conversion Details...5

Packet Format...7

Data System Record 9

Keyboard Commands 10

Conclusion..10

Figures 11

Introduction

The Radar Ocean Wave Spectrometer (ROWS) is a NASA airborne remote sensor used to support the development and refinement of satellite radars that measure the ocean surface. Currently, ROWS is being used to measure ocean surface wind and wave information from altitudes of 3-7 km (see Figure 1).

The ROWS scanning and nadir antenna assembly are both fixed to the same mounting plate (see Figures 2 - 6). This plate is hard mounted to a given aircraft with some plane of reference. Since this reference may change from aircraft to aircraft and since the aircraft pitch slowly oscillates during flight, a ROWS pitch controller was proposed to achieve a stable, known reference. Also, a controller for the scanning antenna pointing angle was made a requirement.

History

The original data system and RF section was built in 1975. It first became operational in 1978. In 1984, ROWS participated in numerous underflights for the Shuttle Imaging Radar (SIR-B). 1987 saw work with the Labrador Extreme Wave Experiment (LEWEX). ROWS then fell in a state of disrepair around 1989. Due to the prototype nature of ROWS, many of its components could no longer be serviced or repaired.

John Ward redesigned the entire data acquisition system in 1991. This included a custom designed timing board (by John) along with state-of-art computer hardware and software. I participated in the software design of this system. I wrote assembler code to provide an efficient means of transferring data from one internal subsystem to another. I also wrote the user interface. More details can be found in NASA Technical Memorandum 104560.

Doug Vandemark became project lead for the ROWS around 1993. He improved the system by buying and having me incorporate a faster waveform digitizer board. Other minor changes to the user interface were made.

Initial Work

This work is a continuation of work done by a Robert Swift (Coop from VPI) during the summer of 1998. Robert did the research and purchase of both the stepper motors and driver boards. He also did the 'first cut' design of both a custom controller board (breadboard) and software. Robert was also responsible for getting the appropriate mechanical systems produced by a in-house machinist. In summary, Robert did a fine job! See Robert's report for further details.

My job was to take Robert's hardware, software, and mechanical systems and make them work with our current ROWS data acquisition system. I first drew a formal schematic of Robert's design (with minor changes) and had a wire-wrap board made (see Figure 7). I then designed a power supply system (see Figure 8) using switching supplies. This was need to totally isolate the aircraft ground bus from the controller. I then had all the appropriate cables produced.

As seen in Figures 9, controller input is via RS-232. Essentially, this custom controller accepts two commands…desired pitch axes position and desired scanning antenna angle position. Both positions are in stepper motor counts. It is the job of the ROWS data acquisition system to provide these positions.

Testing the Initial Openloop Stepper Motor Design

The Philips series 92200 digital linear actuator (stepper motor) has a resolution of .001" per step. This actuator is used for scanning antenna angle. The Philips series 92400 digital linear actuator (stepper motor) has a resolution of .002" per step. This actuator is used for pitch angle.

My initial test was to determine how well these motors perform under load over some nominal period of time. I quickly determined they performed poorly due to nonlinear load bearing. Since this is an openloop system, the motors become 'lost' (or lose their reference) after many start and stops. The proper solution to this problem is to add feedback to the controller. This can be done by any number of transducers available on the market.

Our current plan is to make this system work openloop, since that is how the controller software was designed. The remainder of this paper describes how an openloop controller was made to work (given some constraints) with the ROWS data acquisition system.

New Design for Openloop Operation

P-3 Aircraft motion (on autopilot) is typically a very slow rise and fall in pitch angle. This period is generally on the order of minutes. Our initial design adjusted the antenna pitch angle once per second. At this rate, pitch angle error quickly (linearly) increases due to openloop control. Therefore, to reduce the total pitch angle error we lengthened the pitch angle adjustment to once per minute. The total pitch angle error accumulated after an hour (60 pitch angle adjustments) was determined acceptable. Flights longer than one hour should then reset both motors. This reset was made available as a keyboard command within the ROWS data acquisition system. During reset, both motors move to their start positions and then 'zero' step counts.

Current aircraft pitch is determined by a DTI produced INS board found in the ROWS data acquisition system. The negative of this pitch (plus some predetermined offset) is applied or sent to the controller. This effectively nulls the current pitch bias providing a stable reference for both Radar beams.

Keyboard command of the scanning antenna angle was also added to the ROWS data acquisition system. This angle can be adjusted from 12-26 degrees in 0.5 degree increments. To keep openloop errors to a minimum, this adjustment is not made available during data capture. The idea here is to find the antenna angle of choice before data capture. Once data capture commences, the scanning antenna angle controller finds its reference point and makes **one** move to the predetermined position.

Controller Design

A Microchip PIC16C65A 8-bit controller was used for this system. It was chosen for its cost, speed, and ease of programming. The PIC is programmed using Microchip assembler. This program is separated into a foreground routine and a background routine.

Basically, the foreground routine is responsible for responding to any incoming commands from the serial port. It also outputs motor position (in step counts) to this port at a rate of once per second. Finally, this routine sets global variables to the desired motor position.

The background routine is interrupt driven at 512 Hz. Its job is to examine the global variables set by the foreground routine. If those variables are different from those the background routine maintains, the background routine moves the appropriate motor(s) and increments/decrements those background variables until they equal the global variables. In other words, the background routine 'steps' or moves each motor to the desired position set by the foreground routine.

During any given interrupt, only 1 motor is moved. Their movement is then interleaved on an odd/even basis making the step rate for each motor 256 Hz. This was done to keep motor current draw at a minimum. We determined after the fact that this is not always the case because these motors sometimes have a larger current draw when they are idling. In the future, we plan to remove the interleave feature of this controller to reduce complication.

Steps are provided to two stepper motor driver boards designed by UNO. These boards accept a toggling TTL input as step indicator. Motor direction is determined by a TTL input as well. See Figure 3 for a schematic of the controller board. A more complete description of the controller software can be found in the document Robert Swift produced.

Angle Conversion Details

It was decided early on to use the power of the PC to perform conversions from floating point degrees to fixed point step counts required by the controller. The PC then keeps a floating point record of pitch and antenna angle. When an update is needed on the controller, a conversion is made by the PC to step counts. Likewise, when the controller reports back to the PC the motors current positions, another conversion is needed. The

PC simply converts step counts back to floating point degrees. The following steps were made to insure that roundoff errors were kept to a minimum when making the conversions in either direction.

Pitch angle has a range of -5.6 to +5.0 degrees. When the pitch angle motor is in its starting or reference position, the pitch angle is at -5.6 degrees. This corresponds to 0 step counts. When the pitch angle motor is at +5.0 degrees, this corresponds to 1140 step counts.

Scanning antenna angle has a range of +12 to +26 degrees. When the antenna angle motor is in its starting or reference position, the antenna angle is at +26 degrees. This corresponds to 0 step counts. When the antenna angle motor is at +12 degrees, this corresponds to 3006 step counts.

I use simple linear coefficients to convert from one space to another. Below is a table describing them.

Antenna Angle Code Define	Coefficient	Explanation
A_MDTOS	-107.4	Slope Degrees-to-Steps
A_BDTOS	3006	Y-intercept Degrees-to-Steps
A_MSTOD	-0.0093	Slope Steps-to-Degrees
A_BSTOD	28	Y-intercept Steps-to-Degrees

Pitch Angle Code Define	Coefficient	Explanation
P_MDTOS	107.54	Slope Degrees-to-Steps
P_BDTOS	602.26	Y-intercept Degrees-to-Steps
P_MSTOD	0.0093	Slope Steps-to-Degrees
P_BSTOD	-5.6	Y-intercept Steps-to-Degrees

To alleviate round off error when moving the antenna angle in 0.5 degree increments, the following procedure is used. There are (26-12) * 2 or 28 possible desired angles for this controller. The PC keeps track through the following 'C' language conversion.

This is the conversion of antenna angle steps to floating point angle in degrees.

```
temp = (unsigned int)( (float)steps * A_MSTOD + A_BSTOD + 0.5);
angle = (float)temp / 2.0 + 12.0;
```

This is the conversion of floating point angle in degrees to steps.

```
temp = (unsigned int)( (angle - 12.0) * 2.0 );
steps = (unsigned int)( (float)temp * A_MDTOS + A_BDTOS + 0.5 );
```

Since we are not trying to adjust pitch angle in half degree increments, we do not account for round off error. The following 'C' language conversion is used.

```
This is the conversion of pitch angle steps to floating point angle in degrees.

angle = (float)steps * P_MSTOD + P_BSTOD;
```

A normal addition of 0.5 is made for the following conversion to account for integer truncation error due to type casting.

```
This is the conversion of floating point pitch angle in degrees to steps.

steps = (unsigned int)( (float)angle * P_MDTOS + P_BDTOS + 0.5);
```

Packet Format

Data moves **to** and **from** the controller in a packet format. These packets are simply 2 bytes long and have the following format.

Packet format to the controller		
Motor1 (Pitch angle)	00bxxxxx	1xxxxxxx
Motor2 (Antenna angle)	01bxxxxx	1xxxxxxx

Looking at the packet format 'to the controller', you can see the left most bit or bit 7 of each byte is the packet byte identifier. '0' in this position indicates this is the first byte of a 2 byte packet. '1' in this position indicates this is the second byte of a 2 byte packet. The next bit (bit 6) of the first byte is the motor identifier. '0' in this position indicates this is a packet from Motor1. '1' in this position indicates this is a packet from Motor2.

Bit 5 is a 'reset' bit. When this bit is '1', it acts as a signal to the controller to reset the given motor. A reset moves the motor to its start position bumping up against a mechanical stop. The controller count is also reset to '0'. Finally, data is contained in 12 bits indicated by 'x'. The Lsb of this 12 bit word is found in bit 0 of byte 2. The Msb of this 12 bit word is found in bit 4 of byte 1.

Packet format from the controller		
Motor1 (Pitch angle)	00xxxxxx	10xxxxxx
Motor2 (Antenna angle)	01xxxxxx	11xxxxxx

The packet format 'from the controller' is slightly different from the incoming packet. Bit 7 is again the packet byte indicator with the same conversion as before. Bit 6 is a redundant motor identifier. '0' in this position (from either byte) indicates this is a packet from Motor 1. '1' in this position (from either byte) indicates this is a packet from Motor 2. Redundancy is needed as insurance in case the ROWS data acquisition system 'drops' or looses a byte. This way we always know which byte of the packet we have and which Motor it came from.

Bits 0-5 of each byte are data bits. Again, data is contain in a 12 bit word. The Lsb of this 12 bit word is found in bit 0 of byte 2. The Msb of this 12 bit word is found in bit 5 of byte 1.

Data System Record

The current ROWS data system record looks like:

Byte Start	LSB Followed by MSB	Word Value
0	Header Type	0
2	Header Length	58
4	New ROWS Waveform Type	16
6	New ROWS Waveform Length	1800
8	PC Tick Time Type	2
10	PC Tick Time Length	4
12	Radar Range Type	3
14	Radar Range Length	2
16	High Accuracy Time Type	4
18	High Accuracy Time Length	3
20	New Shaft Angle Type	13
22	New Shaft Angle Length	2
24	LIGHT INS Type	19
26	LIGHT INS Length	100
28	Keyboard Attenuator Type	14
30	Keyboard Attenuator Length	2
32	DAS 1600 A/D Type	18
34	DAS 1600 A/D Length	16
36	Signetec A/D Temp. Type	17
38	Signetec A/D Temp. Length	1
40	Pitch Angle Offset Type	20
42	Pitch Angle Offset Length	4
44	Pitch Angle Type	21
46	Pitch Angle Length	4
48	Antenna Angle Type	22
50	Antenna Angle Length	4
52	Interrupt Count Pitch Type	23
54	Interrupt Count Pitch Length	4
56	End of Header Type	65535
58-2004	The Above Data	

From above, we see that each record contains a 58 byte header followed by 1947 bytes of data. Maintaining a header for each record ensures we always know what is contained in the data portion of the record. This is important since the ROWS data record changes on occasion when either new sensors or different data types are added or removed from the system.

Keyboard Commands

I	Initialize System. Resets all starting variables and readies tape for streaming input. Freezes system for approximately 60 seconds.
ESC	Stop data system and return to DOS prompt.
G	Start recording (must Initialize System first).
S	Stop recording.
+	Move position of range gate forward in time.
-	Move position of range gate backward in time.
U	Increase attenuation of altimeter antenna amplifier.
D	Decrease attenuation of altimeter antenna amplifier.
J	Increase attenuation of scanning antenna amplifier.
K	Decrease attenuation of scanning antenna amplifier.
N	Increase scanning antenna incidence angle by ½ degree (will **NOT** work while recording).
M	Decrease scanning antenna incidence angle by ½ degree (will **NOT** work while recording).
Z	Move both stepper motors to start positions (will **NOT** work while recording). Freezes system for approximately 35 seconds.

Conclusion

Due to many factors out my control, this system has not flown on an aircraft this fall. It will hopefully fly the winter or spring of 1999 on the Orion P3 or C-130. Other changes (hardware and software) are planned for this system. This includes updating the recording media from Exabyte 8mm tape to something else more reliable. We would also like to provide feedback to each controller creating a closed-loop system. This will require hardware and software updates.

Finally, if time and money present themselves, we would like to update the data acquisition system. Currently, it operates in a DOS environment with a custom written user interface. We are looking towards operating systems that are both robust and support a wide array of recording devices. Linux looks like a good solution given the wide user support and availability of many device drivers (both old and new). This operating system also has a real-time kernel which has proven to be very robust for another in-house project.

Figures

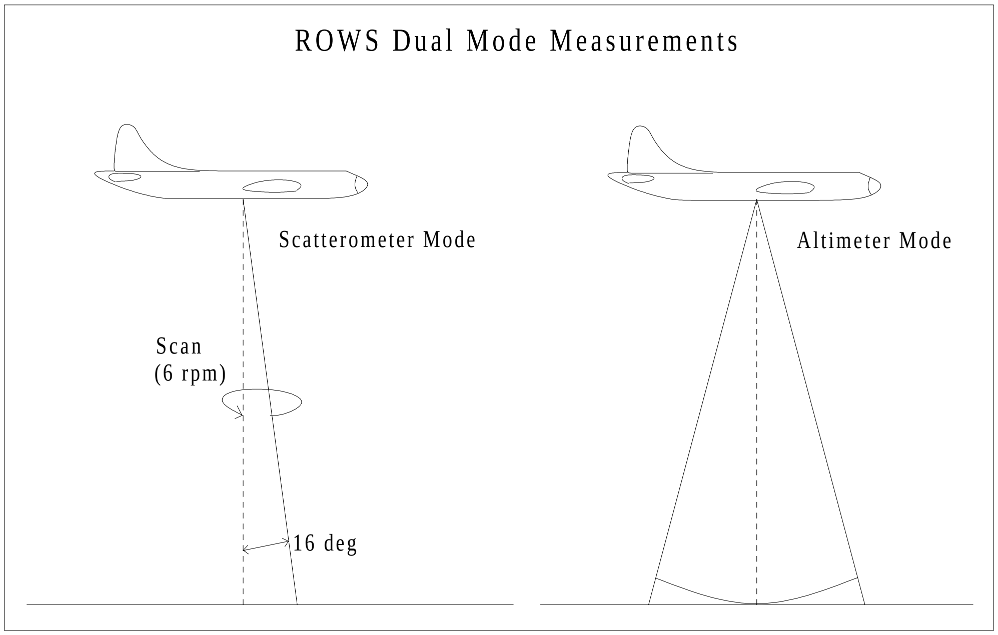


Figure 1 - ROWS Modes

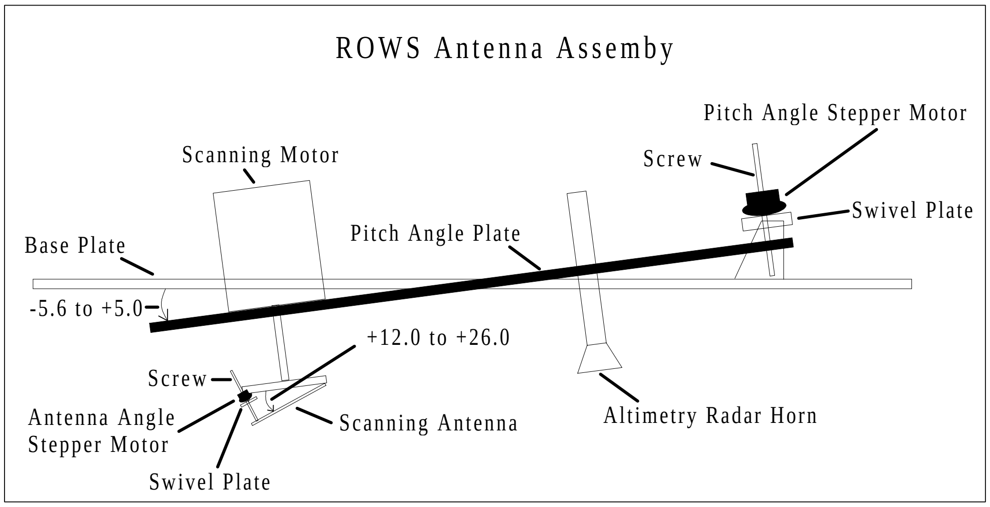


Figure 2 - ROWS Antenna Assembly

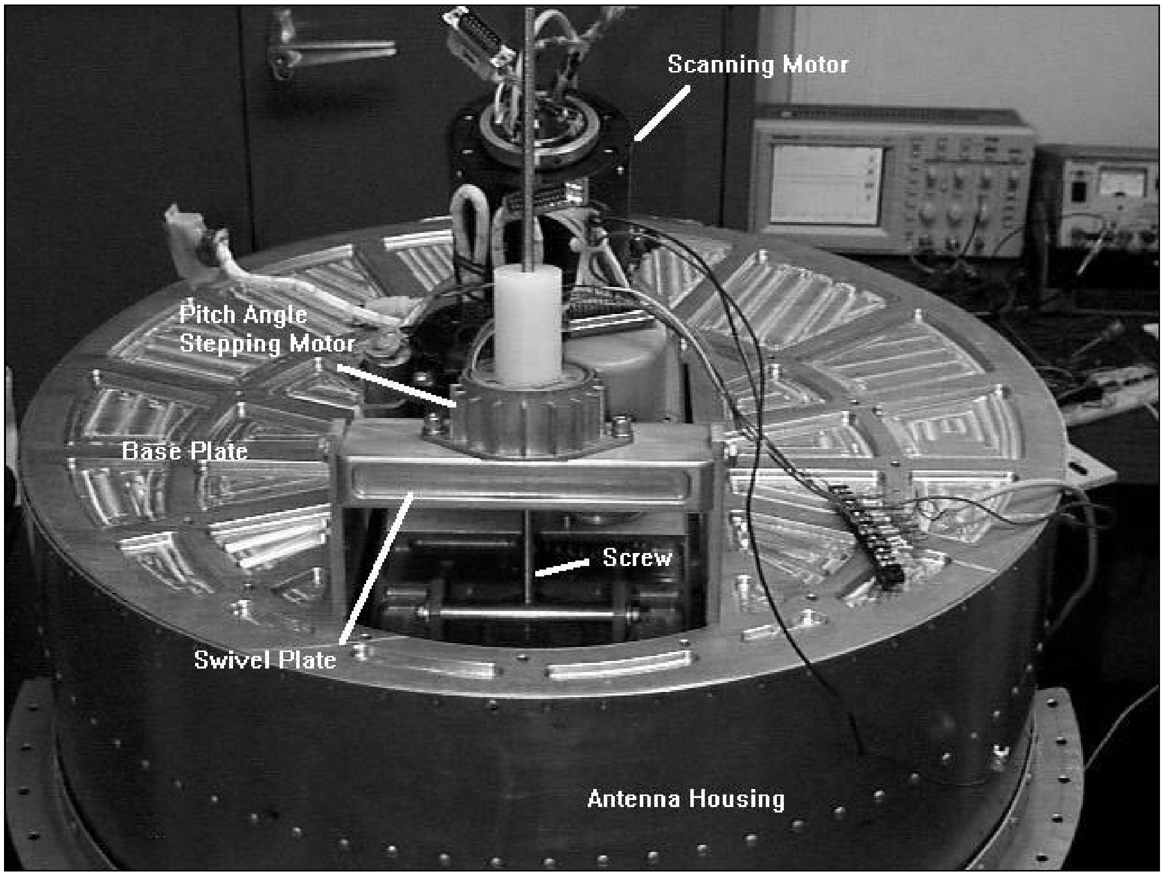


Figure 3 - Outside Endview of Antenna Assembly

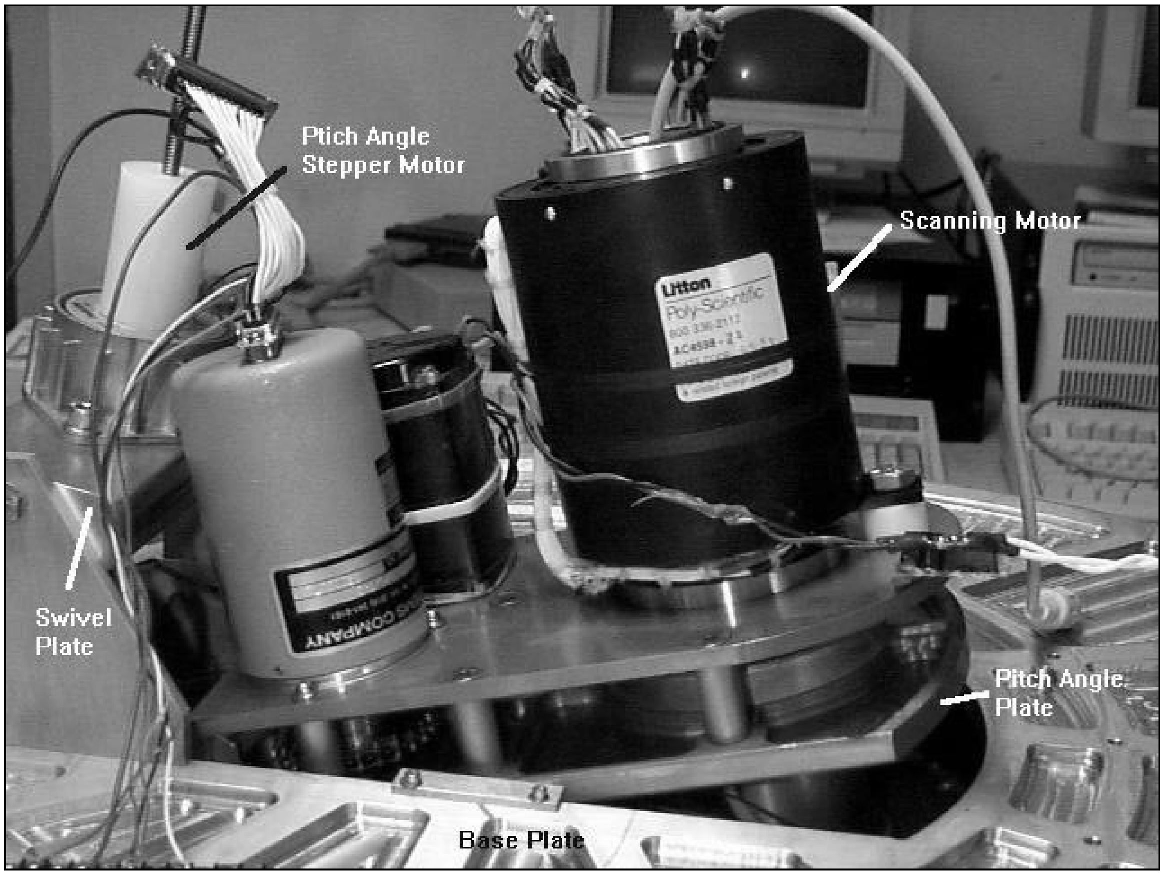


Figure 4 - Outside Sideview of Antenna Assembly

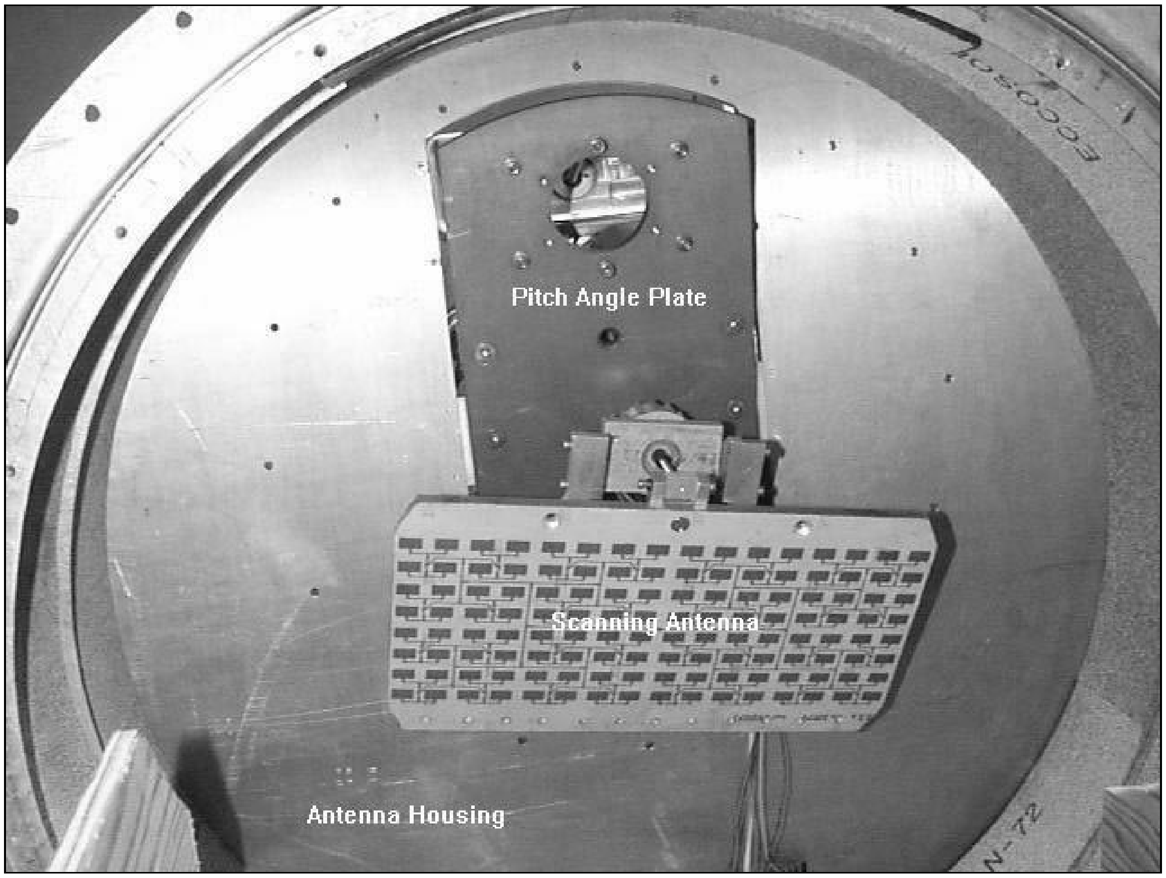


Figure 5 - Inside Antenna Assembly

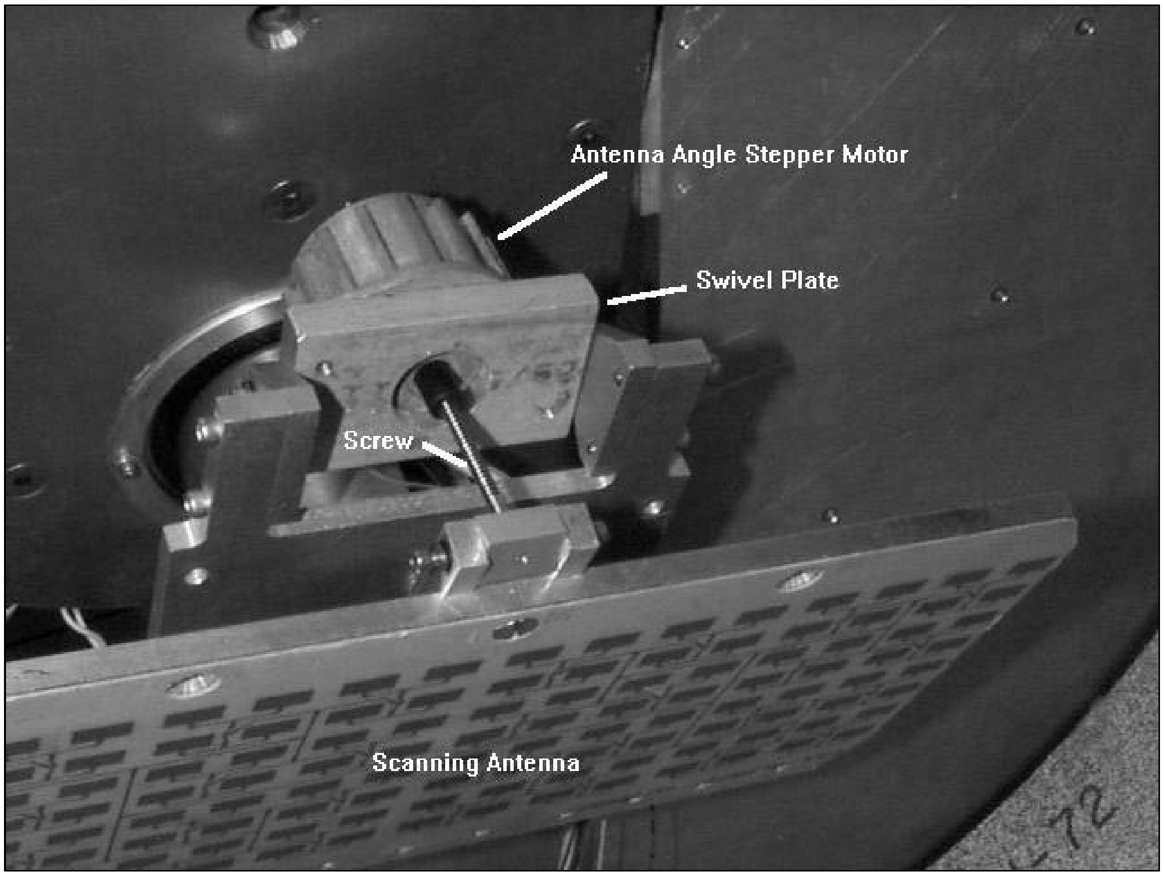
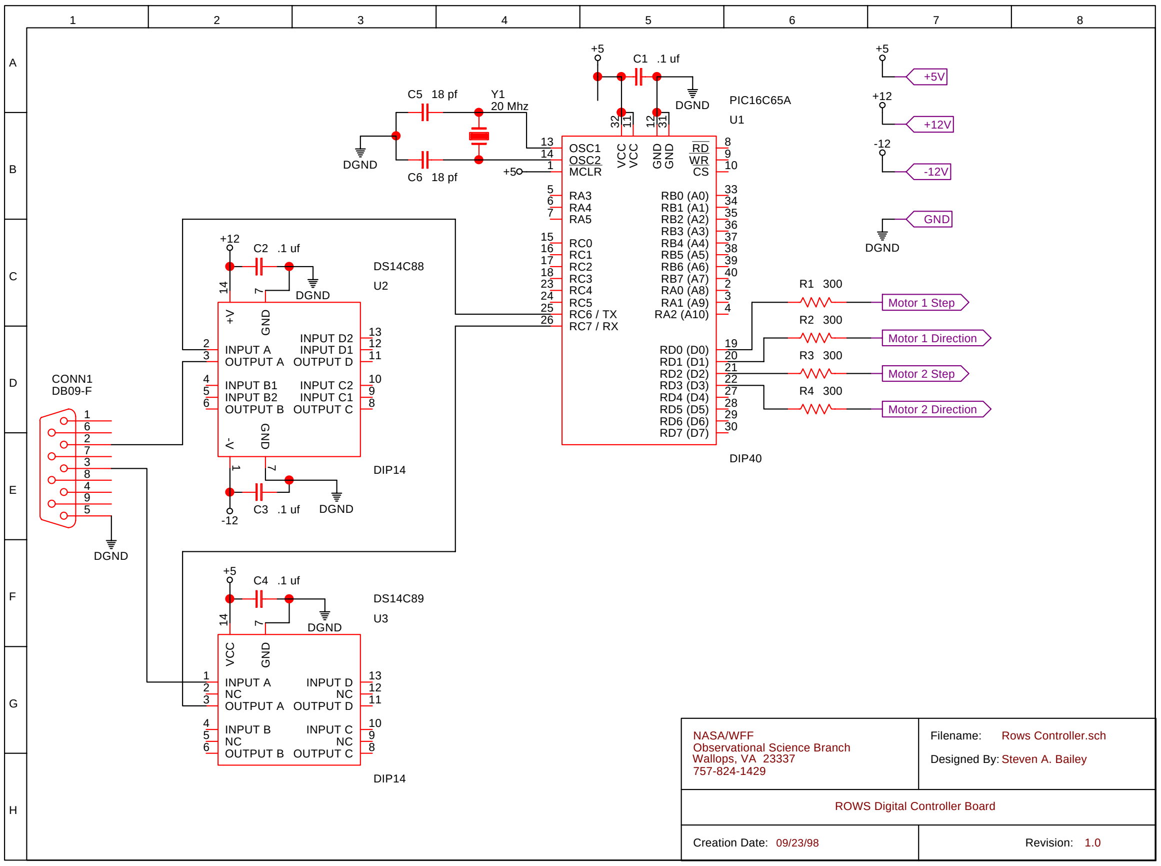


Figure 6 - Close Inside Antenna Assembly



1 2 3 4 5 6 7 8
A B C D E F G H
PIC16C65A
U1
DIP40
OSC1
OSC2
MCLR
RA3 RA4 RA5
RC0 RC1 RC2 RC3 RC4 RC5 RC6 / TX RC7 / RX
RB0 (A0) RB1 (A1) RB2 (A2) RB3 (A3) RB4 (A4) RB5 (A5) RB6 (A6) RB7 (A7) RA0 (A8) RA1 (A9) RA2 (A10)
RD0 (D0) RD1 (D1) RD2 (D2) RD3 (D3) RD4 (D4) RD5 (D5) RD6 (D6) RD7 (D7)
RD WR CS
VCC VCC GND GND
C1 .1 uf
C5 18 pf
C6 18 pf
Y1 20 Mhz
DGND
+5 +12 -12
+5V +12V -12V GND
R1 300
R2 300
R3 300
R4 300
Motor 1 Step
Motor 1 Direction
Motor 2 Step
Motor 2 Direction
DS14C88
U2
DIP14
C2 .1 uf
C3 .1 uf
INPUT A OUTPUT A INPUT B1 INPUT B2 OUTPUT B
INPUT D2 INPUT D1 OUTPUT D INPUT C2 INPUT C1 OUTPUT C
+V GND -V
CONN1
DB09-F
DS14C89
U3
C4 .1 uf
INPUT A NC OUTPUT A INPUT B NC OUTPUT B
INPUT D NC OUTPUT D INPUT C NC OUTPUT C
VCC GND
NASA/WFF
Observational Science Branch
Wallops, VA 23337
757-824-1429
Filename: Rows Controller.sch
Designed By: Steven A. Bailey
ROWS Digital Controller Board
Creation Date: 09/23/98
Revision: 1.0

Figure 7 - Controller Schematic

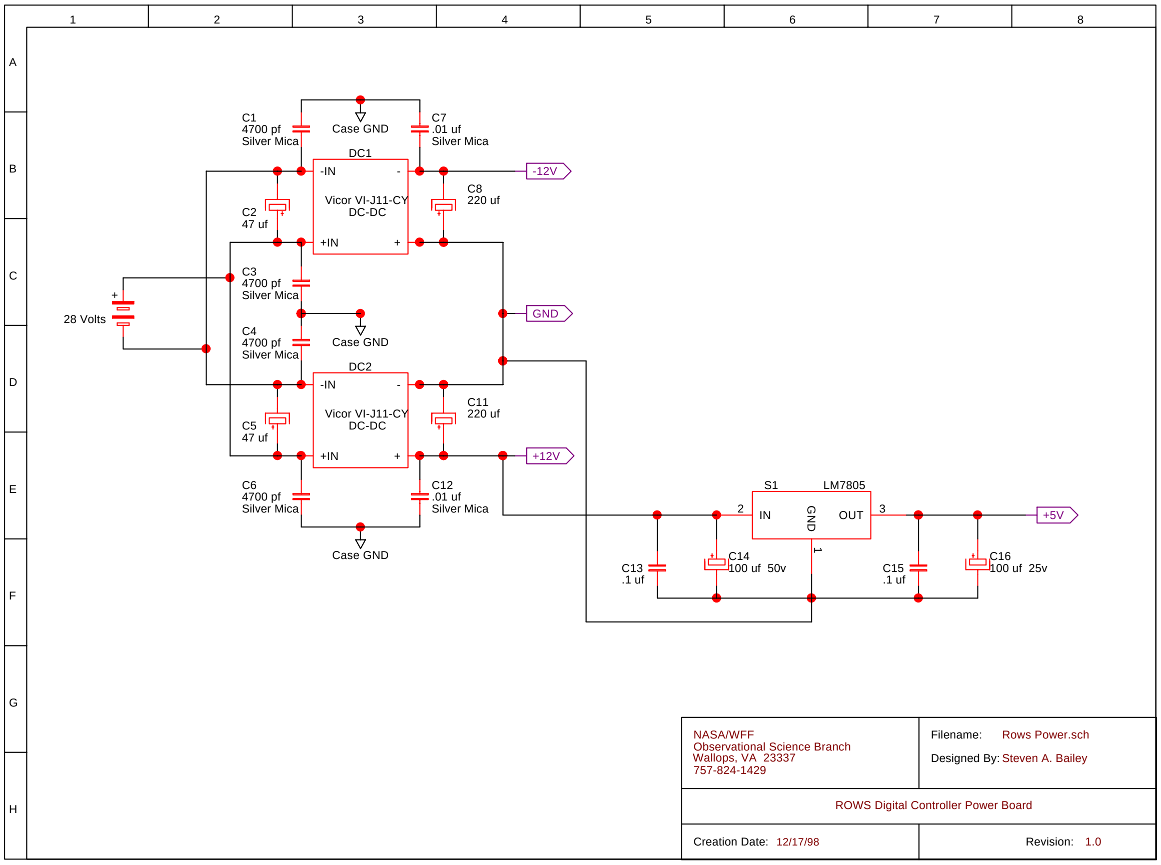


Figure 8 - Power Supply

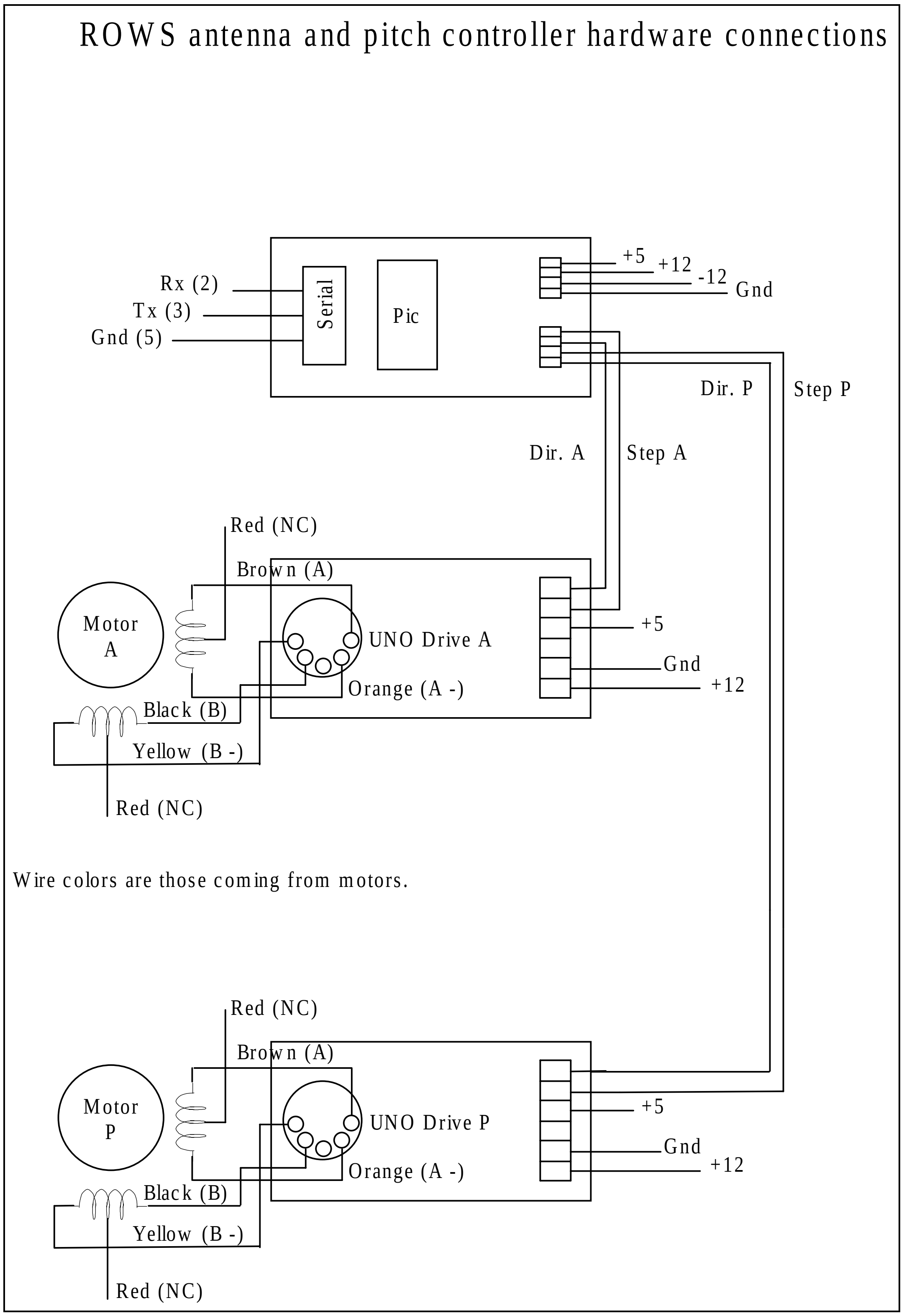


Figure 9 - Hardware Connections